PIMA-Logic: A Novel Processing-in-Memory Architecture for Highly Flexible and Energy-Efficient Logic Computation

Shaahin Angizi, Zhezhi He and Deliang Fan
Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816
{angizi,elliot.he}@knights.ucf.edu,dfan@ucf.edu

ABSTRACT

In this paper, we propose PIMA-Logic, as a novel Processing-in-Memory Architecture for highly flexible and efficient Logic computation. Instead of integrating complex logic units in cost-sensitive memory, PIMA-Logic exploits a hardware-friendly approach to implement Boolean logic functions between operands either located in the same row or the same column within entire memory arrays. Furthermore, it can efficiently process more complex logic functions between multiple operands to further reduce the latency and power-hungry data movement. The proposed architecture is developed based on Spin Orbit Torque Magnetic Random Access Memory (SOT-MRAM) array and it can simultaneously work as a non-volatile memory and a reconfigurable in-memory logic. The device-to-architecture co-simulation results show that PIMA-Logic can achieve up to 56% and 31.6% improvements with respect to overall energy and delay on combinational logic benchmarks compared to recent Pinatubo architecture. We further implement an in-memory data encryption engine based on PIMA-Logic as a case study. With AES application, it shows 77.2% and 21% lower energy consumption compared to CMOS-ASIC and recent RIMPA implementation, respectively.

ACM Reference format:
Shaahin Angizi, Zhezhi He and Deliang Fan. 2018. PIMA-Logic: A Novel Processing-in-Memory Architecture for Highly Flexible and Energy-Efficient Logic Computation. In *Proceedings of DAC '18: The 55th Annual Design Automation Conference 2018, San Francisco, CA, USA, June 24–29, 2018 (DAC '18),* 6 pages.
DOI: 10.1145/3195970.3196092

1 INTRODUCTION

In the last two decades, Processing-in-Memory (PIM) architectures, as a potentially viable way to solve memory wall challenge in conventional Von-Neumann computer system (e.g., long memory access latency, significant congestion at I/Os, limited memory bandwidth and huge leakage power), have been well explored [1]. The key concept behind PIM is to embed logic units within memory to better exploit the external and internal memory bandwidth. This can lead to remarkable saving in data communication energy and latency besides providing an inherent in-memory parallelism for data-processing. An ideal PIM architecture should be capable of performing bit-wise operations used in a wide spectrum of applications [2]. The proposals for exploiting DRAM- [3] and SRAM-based [4] PIM architectures can be found in recent literature. However, they encounter inevitable drawbacks such as high leakage power or initial data overwritten that hinder their further processing.

This topic has become even more intriguing by emerging non-volatile memory (NVM) technology, such as Phase Change Memory (PCM) [5] and resistive RAM (ReRAM) [1]. In early 2016, Everspin announced 256Mb MRAM chips based on Magnetic Tunnel Junction (MTJ), as a spintronic paradigm, with interface speed similar to DRAM and was planning 1Gb chips in near future [6]. Toshiba and SK Hynix co-developed a 4-Gbit MRAM chip prototype shown in IEDM 2016 [7]. Thus, MRAM technology becomes a promising high performance NVM candidate for both last level cache and main memory. Hence, PIM in the context of different NVMs, specially using spintronic devices, without sacrificing memory capacity can open a new way to efficient computing paradigm.

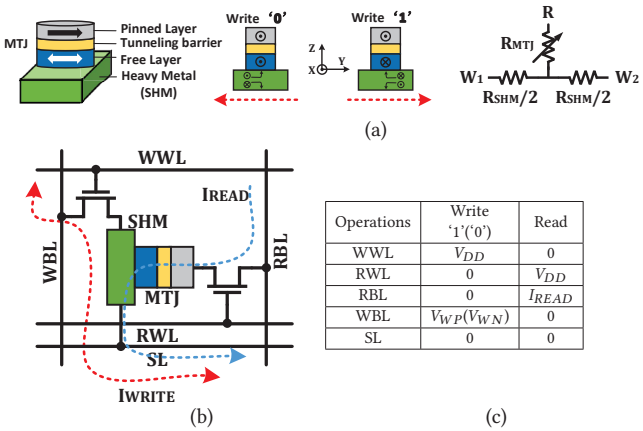
Representative works of such PIM architecture include Pinatubo [2] as a general architecture capable of doing bulk bit-wise operations, MPIM [8] as a multi-purpose ReRAM-based PIM, PRIME [1] and ISAAC [9] as dot-product engines for Neural Network acceleration based on ReRAM, RIMPA [10] as a threshold logic PIM architecture based on Domain Wall-RAM (DW-RAM) and STT-MRAM based PIM in [11]. All these architectures can be reconfigured to memory and computing modes where modified Sense Amplifiers (SA) typically play major role in performing row-wise or column-wise in-memory logic.

However, these recent PIM designs have a few limitations. First, none of these designs can perform computing (Boolean logic functions) between any two bits irrespective of their locations in the memory array. As a matter of fact, processing data (operands) can be stored in different memory locations with distinct physical addresses. Therefore, existing bit-wise PIM schemes unavoidably impose multi-cycle operations to align operands in the same column [2, 4] or row [10] to process data within memory. For instance, RIMPA [10]/Pinatubo [2] require at least 2 cycles (read/write) to line operands in the same row/column to realize a 2-input in-memory AND function. Thus, this issue is a very important un-addressed topic in previously reported PIM architectures [4]. Second, current PIM schemes unavoidably rely on external processing unit for performing more complex logic operations, otherwise PIM's performance degradation would be considerable due to multi-cycle operations. For instance, addition can be more efficiently performed by ALU rather than PIM platforms.

To mitigate associated PIM challenges, we propose PIMA-Logic, as a novel PIM architecture capable of performing complex logic computations. Our contributions in this work can be briefly listed as: (1) We design an efficient and reconfigurable PIM architecture based

DAC '18, San Francisco, CA, USA

DOI: 10.1145/3195970.3196092



Operations	Write '1'('0')	Read
WWL	V_{DD}	0
RWL	0	V_{DD}
RBL	0	I_{READ}
WBL	$V_{WP}(V_{WN})$	0
SL	0	0

Figure 1: (a) SOT-MRAM device structure, (b) Schematic and (c) biasing conditions of SOT-MRAM bit-cell.

on SOT-MRAM with enhanced and modified peripheral circuitry; (2) We pave a new way to further push the boundaries of PIM so that bit-wise computations can be performed between operands with flexible locations (either in the same row or column) within memory. This further reduces the latency and power concerning state-of-the-art PIM hardware; (3) We propose Resistive Unit (RUnit) to PIMA-Logic to handle and accelerate complex in-memory logic computations by realizing a majority logic after memory sense amplifiers; (4) We implement an in-memory data encryption engine design based on PIMA-Logic as a case study to further explore its superior performance compared to counterparts.

2 PIMA-LOGIC

2.1 Device

Fig. 1a shows a Spin-Orbit Torque Magnetic Random Access Memory (SOT-MRAM) device structure with the composite structure of spin Hall metal (SHM) and MTJ. Here, the resistance of MTJ with parallel magnetization in both magnetic layers (data-'0') is lower than that of MTJ with anti-parallel magnetization (data-'1'). To program free-layer magnetization, flow of charge current (±y) through SHM (Tungsten, $\beta - W$ [12]) will cause accumulation of opposite directed spin on both surfaces of SHM due to spin Hall effect [13]. Thus, a spin current flowing in ±z is generated and further produces spin-orbit torque (SOT) on the adjacent free magnetic layer, causing switch of magnetization. The bit-cell structure of 2T1R SOT-MRAM and its biasing conditions are shown in Fig. 1b and 1c, respectively. In this work, the magnetization dynamics of the free ferromagnetic (FM) layer is modeled by LLG equation with STT term and SHE term, as used in [14]. Note that the ferromagnets in MTJ have In-plane Magnetic Anisotropy (IMA) in x-axis [13]. With the given thickness (1.2nm) of the tunneling layer (MgO), the Tunnel Magneto-Resistance (TMR) of the MTJ is ~ 168.5%.

2.2 Circuit and Architecture

2.2.1 General Overview. The general memory organization of PIMA-Logic is shown in Fig. 2, inspired by Pinatubo architecture [2] but with significant modifications to enhance overall system performance. Memory chip is basically divided into multiple Banks consisting of multiple Mats. Banks within the same chip typically share I/O, buffer and banks in different chips working in a lock-step manner. The Mats are connected to a Global Row Decoder (GRD)

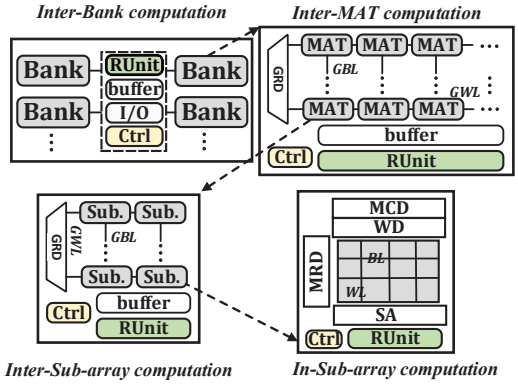


Figure 2: PIMA-Logic Organization.

and a shared row buffer. Each Mat consists of multiple memory sub-arrays. At the lowest memory level, each memory sub-array encompasses Memory Row Decoder (MRD), Memory Column Decoder (MCD), exclusive SA and Write Driver (WD).

According to the physical address of operands within memory, PIMA-Logic Controller (Ctrl) can perform in-memory computation in two levels: 1) Inter-component (i.e. Bank, MAT and Sub-array) computation employing the proposed Resistive Unit (**RUnit**) and 2) In-sub-array computation employing both **Conventional Row-wise PIM** and **RUnit**. Each sub-array can be considered as a computing core in PIMA-Logic architecture. To perform inter-component computation, first operand row is read into devised buffer and accordingly second row is read via Global Bit Line (GBL). Then, computation is accomplished by the RUnit that will be described later.

2.2.2 Sub-Array Architecture. Fig. 3 depicts the PIMA-Logic sub-array architecture and detailed modified peripheral circuitry. The basic sub-array of PIMA-Logic (shown in L.H.S. of Fig. 3) mainly consists of MRD, MCD, WD, SA, multiplexers (MDMUX, GMUX) and RUnit. This architecture can be adjusted by Ctrl unit to work in dual mode that perform both memory read-write and in-memory logic operations. WD component (B) is modified such that can select between data input coming from different sources (i.e. Din-Intra, Din-Inter, RUnit, and GMUX). MRD (C) is modified based on approach proposed in [2] such that two WLs can be selected and sensed simultaneously. Fig. 3 (A) shows the architecture of 2×2 memory array. Each SOT-MRAM cell is associated with the Write Word Line (WWL), Read Word Line (RWL), Write Bit Line (WBL), Read Bit Line (RBL), Source Line (SL) to perform typical memory operations. The peripheral decoders (active-high output) control the activation of current path through the array. WDs are used with the WBLs for providing the required write voltage. A voltage mode SA [13] is connected to the RBL for sensing the total resistance in the selected current path during read or computing mode. Here we discuss about functionality of PIMA-Logic sub-array architecture:

Memory Write: To write a data bit in any of the SOT-MRAM cells, write current should be injected through the heavy metal substrate of SOT-MRAM. To activate this write current path (e.g. for MRAM1 (A)), WWL1 should be activated by MRD and SL1 is grounded, while all the other word lines and source lines are kept deactivated (floating). Data can come from different sources and according to its value, positive (/negative) write voltage (B)

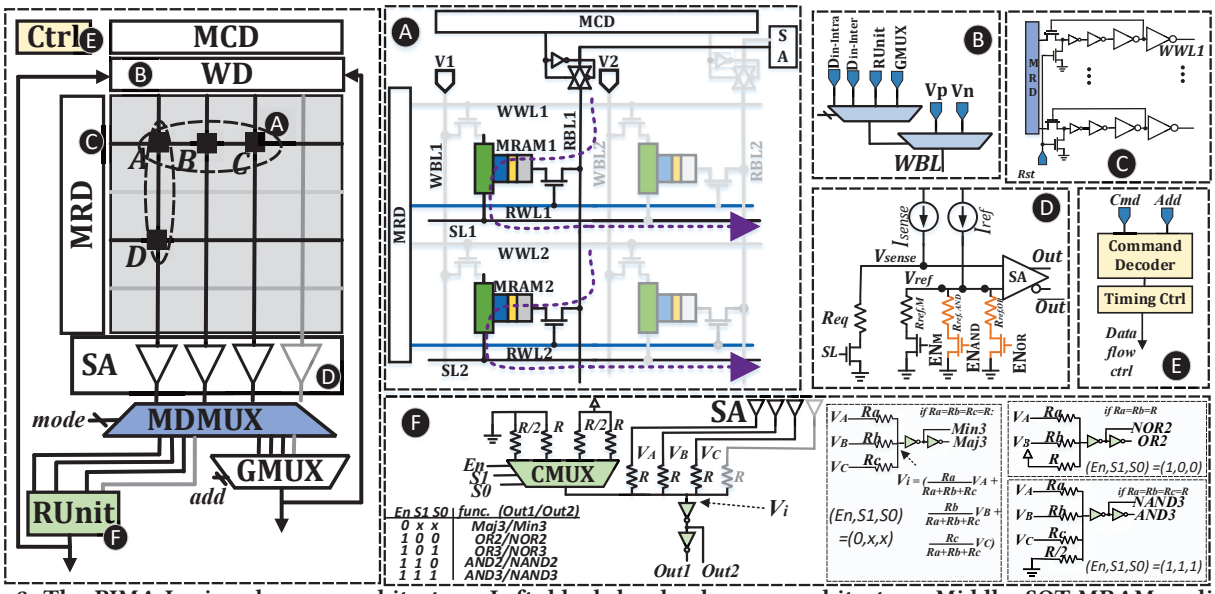


Figure 3: The PIMA-Logic sub-array architecture. Left: block level sub-array architecture, Middle: SOT-MRAM realization, and Right: functional blocks used in sub-array. Glossary: CMUX: Control Mux located at RUnit, Din-Intra: Data input to sub-array, Din-Inter: Data input coming from other sub-arrays, GMUX: Global Mux for interfacing with intra- and inter-subarrays, MDMUX: Mode D-Mux specially designed to select Runit for more complex computations.

should be assigned. Therefore, in order to write '1' (/ '0'), WD sets WBL1 to V_p (/V_n) write voltage. This allows sufficient charge current (~120 μA) flows from V1 to ground (/ground to V1), leading to MTJ resistance in High-R_{AP} (/Low-R_P) encoded as data '1' (/ '0').

Memory Read: For typical memory read, a single memory cell is selected to compare its sense voltage (V_{sense}) with a reference voltage (V_{ref}) by injecting a small sense current (I_{sense}) through the selected SOT-MRAM cell. To activate this read current path for example for MRAM1 (A), RWL1 is activated while SL1 is grounded and all the other word lines and source lines are kept deactivated. MCD activates the RBL1 line to be connected to the SA. Hence, a read current flows from the selected SOT-MRAM cell to ground, generating a sense voltage at the input of SA (D), which is compared with memory mode reference voltage ($V_{sense,P}$<V_{ref}<$V_{sense,AP}$). This reference voltage generation branch is selected by setting the Enable values (EN_M, EN_{AND}, EN_{OR})= (1,0,0). This selection is performed by the command issued by Ctrl unit.

Conventional Row-wise PIM Operation: In this read based computation, every two bits stored in the identical column can be selected and sensed simultaneously as depicted in Fig. 3 (A). To activate the computing current path (as shown in Fig. 3 (A)), RWL1 and RWL2 are activated by MRD while SL1 and SL2 are grounded and all the other word/source lines are kept deactivated. Then, the equivalent resistance of such parallelly connected SOT-MRAMs (MRAM1 and MRAM2) and their cascaded access transistors are compared with a specific reference by SA. Through selecting different reference resistances (EN_M, EN_{AND}, EN_{OR}), the SA (D) can perform basic in-memory Boolean functions (i.e. AND/NAND and OR/NOR). For AND operation, R_{ref} is set at the midpoint of $R_{AP}//R_P$ ('1','0') and $R_{AP}//R_{AP}$ ('1','1') and for OR operation, R_{ref} is set at the midpoint of $R_P//R_P$ and $R_P//R_{AP}$.

Fig. 4a depicts the transient simulation result of the sense circuit under the 2ns period clock signal (CLK), which takes the data stored in MRAM1 and MRAM2 as inputs. When CLK is high, SA is in pre-charge phase and the output is reset to '0'. When CLK is low, the sense amplifier is in sampling phase, and generates logic computation result depending on the reference voltage configuration. Furthermore, to validate the variation tolerance of sense circuit, we have performed Monte-Carlo simulation with 100000 trials. A σ = 5% variation is added on the Resistance-Area product (RA_P), and a σ = 10% process variation is added on the TMR (typical MTJ conductance variation[15]). The simulation result of sense voltage (V_{sense}) distributions in Fig. 4b shows the sense margin of such PIM architecture. It will be reduced by increasing the logic fan-in (i.e. number of parallel memory cells). Thus, to avoid read failure (overlapping of V_{sense} distribution), only two fan-in row-wise in-memory logic is used. Note that parallel computing/read within sub-array is implemented by using one SA per bit-line with exact same mechanism.

New Column-wise Majority Operation Using RUnit: We propose RUnit as a low-overhead and highly-efficient solution to process operands located in one memory row either in sub-array or inter-component level. In sub-array level, we have devised a Mode demultiplexer (MDMUX) to switch between conventional PIM mode or proposed enhanced one (PIM+RUnit). As it can be seen in block-level sub-array architecture, the output of each SA is routed to MDMUX. According to the mode selector, output data can be routed to either GMUX or RUnit. The key idea behind RUnit is to realize a majority logic after SAs to further process the data avoiding unnecessary write-back and accelerating in-memory processing. As shown in Fig. 3 (F), in-block circuit design of RUnit consists of n resistors (n = $\#of SA$) that can parallelly contribute to design a voltage divider driving a static CMOS inverter. To do

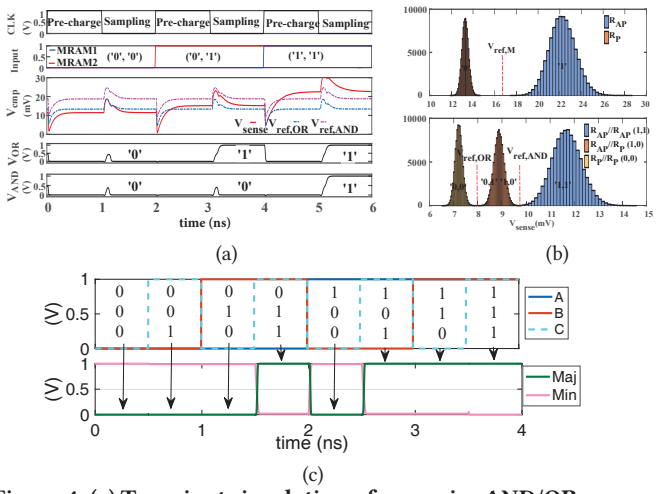


Figure 4: (a) Transient simulation of row-wise AND/OR computation, (b) Monte Carlo simulation result of V_{sense} distribution, (c) Transient simulation of RUnit realizing majority and minority functions when $(En, S1, S2)$**=(0,x,x).**

the computation, MCD is modified (similar to that of MRD) such that it can activate more than one RBL at the same time. As a result, more than one column can be sensed and routed from SAs to RUnit. Considering similar resistance (R), input voltage of inverter (V_i) can be simply derived,

$$V_i = \frac{k.V_{DD}}{w} \tag{1}$$

where k denotes the number of SA outputs carrying V_{DD} and w represents the total number of unit resistors (R) connected to the inverter. Thus, first inverter acts as a threshold detector by amplifying deviation from $\frac{V_{DD}}{2}$ and realizes a minority function. Then, second inverter yields majority function output. In addition to majority/minority function-based computing, RUnit is equipped with CMUX in order to assign different weighted inputs to V_i. This could be used to directly implement column-wise multi-input AND/OR functions. For instance, as shown in Fig. 3 (F), 3-input AND/NAND functions can be efficiently designed by setting $(En, S1, S2)$=(1,1,1). To avoid logic failure due to large number of inputs, causing the V_i to be close to $\frac{V_{DD}}{2}$, we have limited the number activated columns to three. However, when CMUX is deactivated ($(En, S1, S2)$=(0,x,x)), our simulations showed that up to five columns can be reliably sensed and computed. To better explain the proposed circuit, let's assume A, B and C operands are located in the way shown in L.H.S of Fig. 3. For calculating the minority and majority functions in a single cycle, 3 RBLs are activated simultaneously and sensed. CMUX is set by Ctrl to $(En, S1, S2)$=(0,x,x). We expect the result of sensed RBLs to be 0 after second inverter if at least two of the three SA outputs are '0' ($k = 0, 1$), and the result to be V_{DD}, if at least two of three SA outputs are carrying '1'($k = 2, 3$), in this way:

$$\begin{cases} V_i < \frac{V_{DD}}{2} \Rightarrow V_{Out1} = 0, & k = 0, 1 \\ V_i > \frac{V_{DD}}{2} \Rightarrow V_{Out1} = V_{DD}, & k = 2, 3 \end{cases} \tag{2}$$

Transient simulation result of RUnit performing minority/majority functions is depicted in Fig. 4c considering A, B and C as inputs. It is noteworthy that considering data that is not aligned neither in a same row nor column, PIMA-Logic's column-wise and row-wise operations need more than one cycle to line data in either same row or column to perform the computation.

Fig. 5 intuitively depicts performing some simple Boolean functions within PIMA-Logic compared to Pinatubo [2]. As it can be seen, A and B operands can be processed (AB) efficiently in one single cycle regardless of their physical address using conventional row-wise PIM operation (Conv. PIM) and column-wise operation using RUnit of PIMA-Logic. However, similar function is implemented in 3 cycles using Pinatubo when operands are not aligned in one column . This can be further explored while computing more complex logic functions. As shown, majority function ($AB+AC+BC$) can be computed in one single cycle using PIMA-Logic, however Pinatubo needs more than 10 cycles to perform such function.

In inter-component level, we consider two RUnit per component (Sub-array, MAT, Bank). If the operands are in different subarrays (/MATs/Banks) within one MAT (/Bank/memory chip), PIMA-Logic performs inter-component operations employing RUnit added on the row buffer. The first operand row is read into devised buffer and accordingly second operand is read via GBL. After computation, the final result is latched in the row buffer.

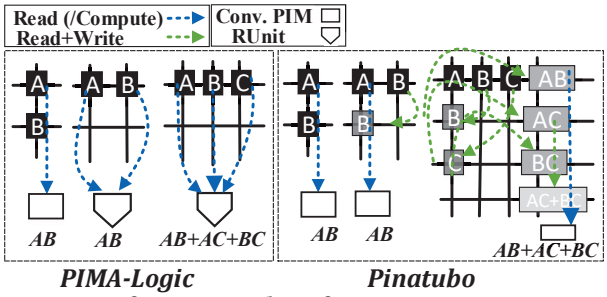


Figure 5: Performing Boolean functions using PIMA-Logic and Pinatubo [2].

For further exploration, we perform two experiments to thrive the superiority of PIMA-Logic compared to two recent PIM architectures (i.e. RIMPA [10] and Pinatubo [2]). Table 1 tabulates the synthesis of 13 standard functions [16], to represent all 256 possible 3-variable Boolean functions, utilizing different platforms. To perform an impartial comparison, we first assume that initial physical addresses for all operands are either in the same column (CS) or same row (SR). Based on Table 1, PIMA-Logic can show up to 36.5% and 43.9% improvement in terms of average number of cycles compared to RIMPA and Pinatubo, respectively for processing 13 functions with SR condition. In second experiment, data is randomly-distributed (RD) in memory and we perform the computation. In this case, PIMA-Logic can show up to 43.1% and 50.8% improvements compared to RIMPA and Pinatubo, respectively.

As an instance of combinational logic circuits, we show the realization of a full-adder within PIMA-Logic in Fig. 6. Assuming A, B and C are initially located in a memory row, Carry output (C_{out}) is generated in a single cycle (see function 9 in Table 1), accordingly Sum can be generated as $Sum = M5(A, B, C, \overline{C_{out}}, \overline{C_{out}})$ after three cycles. We generalize the idea by implementing an efficient in-memory 4-bit ripple Carry Adder (RCA). As shown, column-wise computation employing RUnit could be adopted to realize such complex circuit very efficiently.

Table 1: Synthesis comparison of the 13 standard functions.

No.	Standard Function	RIMPA[10]			Pinatubo[2]			PIMA-Logic		
		SC	SR	RD	SC	SR	RD	SC	SR	RD
1	F=AB'C	7	5	5	5	7	9	5	3	3
2	F=AB	3	1	3	1	3	5	1	1	1
3	F=A'BC+A'B'C'	19	15	17	15	19	17	9	11	9
4	F=A'BC+AB'C'	13	17	13	13	17	17	11	11	9
5	F=A'B+BC'	13	9	11	9	13	11	9	7	7
6	F=AB'+A'BC	11	11	11	11	11	15	9	7	7
7	F=A'BC+ABC'+A'B'C'	21	25	21	21	25	25	13	13	11
8	F=A	1	1	1	1	1	1	1	1	1
9	F=AB+BC+CA	5	1	3	7	13	11	5	1	1
10	F=A'B+B'C	9	9	9	9	9	11	9	5	5
11	F=A'B+BC+AB'C'	17	21	19	17	21	21	17	13	11
12	F=AB+A'B'	11	9	9	1	3	3	5	5	5
13	F=ABC'+A'B'C'+AB'C+A'BC	33	29	31	29	31	31	25	19	17
	Total Number of Cycles	162	153	153	139	173	177	119	97	87
	Average Number of Cycles	12.46	11.76	11.76	10.69	13.3	13.6	**9.15**	**7.46**	**6.69**
	Improvement Percentage	26.5%	36.5%	43.1%	14.4%	43.9%	50.8%	-	-	-

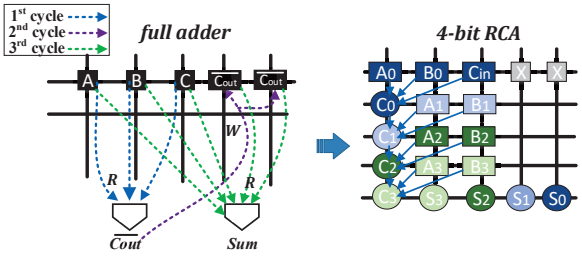


Figure 6: Realization of in-memory full adder and 4-bit RCA.

3 PERFORMANCE EVALUATION

3.1 Logic Performance

To evaluate logic performance of PIMA-Logic, we first need to extract device-to-circuit level data. The simulation is initially carried out in Cadence Spectre with NCSU 45nm CMOS PDK [17]. SOT-MRAM device model of Fig. 1a is used in the circuit simulation. MTJ resistance (R_{MTJ}) is obtained from the NEGF approach [18], while the heavy metal resistance (R_{HM}) is calculated based on the resistivity and device dimension. Accordingly, a logic netlist in Berkeley Logic Interchange Format (.blif) is fed into ThrEshold Logic Synthesizer (TELS) [19] to obtain synthesized logic networks. Meanwhile, parameters such as fan-in restriction is set up during the synthesis. The synthesized networks are then mapped to PIMA-Logic using an in-house developed Matlab code to assess the performance. Fig. 7 gives ISCAS85 combinational circuit benchmarks implemented using PIMA-Logic, RIMPA [10] and Pinatubo [2]. To have an impartial comparison, Pinatubo, as a general system architecture for non-volatile memories, is implemented with both standard STT-MRAM and identical SOT-MRAM cell depicted in Fig. 1b.

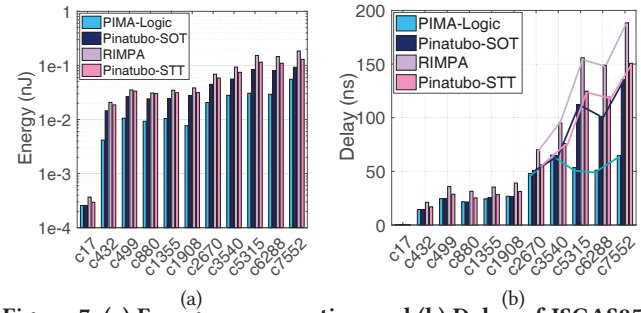


Figure 7: (a) Energy consumption and (b) Delay of ISCAS85 benchmarks mapped to three different PIM architectures (Y-axis in energy plot: Log scale).

As shown, PIMA-Logic exhibits lowest energy and delay compared to the counterparts in different benchmarks. We observe that (1) PIMA-Logic reduces the energy consumption by ~56%, 67% and 74.4% compared to Pinatubo-SOT, Pinatubo-STT and RIMPA, respectively. This considerable improvement mainly comes from proposed logic efficiency and reduced-cycle operations. (2) PIMA-Logic outperforms mentioned PIM architectures with 31.6%, 40% and 52% reduction in delay on different benchmarks. It is worth pointing out that for five more complex benchmarks (i.e. c2670, c3540, c5315, c6288 and c7552), as logic complexity increases, PIMA-Logic can show much better performance compared to the rest.

3.2 Memory Performance

In order to achieve the overall memory performance of PIMA-Logic as an SOT-MRAM based architecture with modified peripheral circuitry, we extensively modified the system level memory evaluation tool NVSim [20] to co-simulate with an in-house developed C++ code based on circuit level results. Table 2 tabulates and compares the memory performance (Write (W)/Read (R)) of PIMA-Logic based on SOT-MRAM with three different memory candidates for a sample 4MB memory chip in 45nm process node.

Table 2: Memory Model Comparison

	SRAM 4MB		DRAM 4MB		Standard STT-MRAM 4MB		PIMA-Logic 4MB	
Metrics	W	R	W	R	W	R	W	R
Latency (ns)	1.07	0.9	2.7	2.4	10.2	1.08	1.31	1.1
Dynamic Energy (pJ)	297.4	312.5	967	1483	368.5	232.2	302.7	275.4
Leakage Power (mW)	5258		585.4		744.2		782.5	
Area (mm^2)	10.544		6.504		5.963		6.164	

In our simulations, we follow iso-capacity constraint where similar memory capacity is used for all the candidates. As expected, magnetic memories and DRAM show smaller area overhead compared to SRAM. PIMA-Logic memory model imposes 41.5% less area compared to SRAM with same memory configuration. Besides, the magnetic memory models save a lot of leakage power compared to SRAM due to their non-volatility nature. PIMA-Logic also outperforms other candidates in terms of dynamic energy owning to its low write voltage (~400mV for'1' and ~-320mV for'0'). Albeit, SOT-MRAM array design used in PIMA-Logic improves the write energy and latency compared to standard STT-MRAM and DRAM, all magnetic candidates have shown longer write latency compared to SRAM due to longer write period of magnetic storage devices.

3.3 Area Overhead

Fig. 8 shows break-down of area overhead resulted from add-on hardware to memory chip. Our experiments show that totally PIMA-Logic imposes 1.4% area overhead to original memory die, where Pinatubo [2] and RIMPA [10] incur 0.9% and 17% area overhead, respectively. It can be seen that modified controller and drivers contribute more than 50% of this area overhead in a Bank.

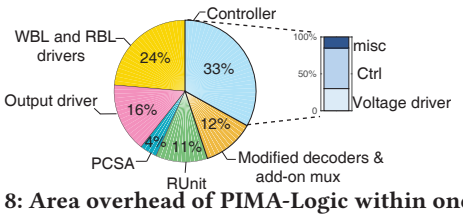


Figure 8: Area overhead of PIMA-Logic within one Bank

4 APPLICATION: DATA ENCRYPTION

In this section, we evaluate the performance of PIMA-Logic as an in-memory data encryption engine for Advanced Encryption Standard (AES) algorithm [21]. AES works on the standard input length of 16 bytes data organized in a 4×4 matrix (state matrix) while using 3 different key lengths (128, 192, 256 bits). For 128-bit key length, AES encrypts the input data after 10 rounds of consecutive transformations, i.e. SubBytes, ShiftRows, MixColumns, and AddRoundKey (Fig. 9a).

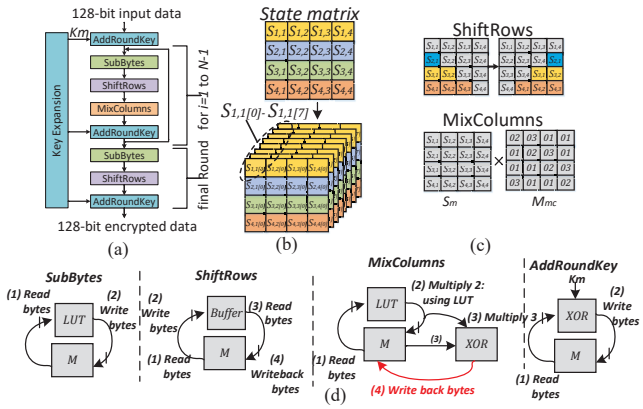


Figure 9: (a) AES block diagram, (b) Data organization, (c) AES's ShiftRows and MixColumns transformation, (d) Mapping of four AES transformations to PIMA-Logic.

To facilitate working with input data as depicted in Fig. 9b, each byte in input data is distributed into 8-bit. So, 8 memory arrays are filled by 4×4 bitmatrices. Mapping of four AES transformations to PIMA-Logic is shown in Fig. 9d. For evaluation of AES performance in general purpose processor (GPP), we have used similar method in [22] at 2GHz. AES C code is extracted from [21] and compiled, then cycle-accurate architecture simulator gem5 [23] is employed to take AES binary and accordingly system level processor power evaluating tool McPAT [24] is used to estimate power dissipation. For evaluation of AES in CMOS ASIC (1.133GHz), Synopsys Design Compiler tool is used. Here, the performance for all the platforms are listed in Table 3 in 32nm technology. For fair comparison, we have done fixed-voltage scaling of the results obtained from of our work to 32nm by using the appropriate scaling factor- which is $(1/S^2)$ for area and $(1/S)$ for energy [25], here S= L/32nm, where L=45nm. The device-to-architecture co-simulation results show that PIMA-Logic can achieve 77.2% and 21% lower energy consumption compared to CMOS-ASIC and RIMPA implementations, respectively at 30MHz. Furthermore, PIMA-Logic occupies ~ 3× less area compared to baseline DW-AES.

Table 3: Comparison of 128-bit AES implementations.

Platforms	Energy (nJ)	Cycles	Area (μm^2)
GPP [21]	460	2309	2.5e+6
ASIC [26]	6.6	336	4400
CMOL[27]	10.3	470	320
Baseline DW [22]	2.4	1022	78
Pipelined DW [22]	2.3	2652	83
Multi-issue DW [22]	2.7	1320	155
RIMPA [10]	1.9	1084	92
PIMA-Logic	1.5	872	27

5 CONCLUSION

In this paper, we proposed PIMA-Logic as a novel memory architecture to further push the boundaries of processing-in-memory so that complex bit-wise computations can be performed between locally-flexible operands (either in the same row or in the same column) within memory. The device-to-architecture simulation results show that PIMA-Logic can achieve up to 56% and 31.6% improvements with respect to overall energy and delay on large scale logic benchmarks compared to well-designed Pinatubo equally-implemented with SOT-MRAM arrays.

ACKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation under Grant No. 1740126 and Semiconductor Research Corporation nCORE

REFERENCES

[1] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, vol. 43, 2016.
[2] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *2016 53nd DAC*. IEEE, 2016.
[3] V. Seshadri and O. Mutlu, "Simple operations in memory to reduce data movement," *Advances in Computers*, 2017.
[4] S. Aga *et al.*, "Compute caches," in *HPCA, 2017 IEEE International Symposium on*. IEEE, 2017, pp. 481–492.
[5] B. C. Lee *et al.*, "Architecting phase change memory as a scalable dram alternative," in *ACM SCAN*, 2009.
[6] "Everspin stt. 2016. url: https://www.everspin.com/news 256mb-perpendicular-spin-torque-mram."
[7] S.-W. Chung *et al.*, "4gbit density stt-mram using perpendicular mtj realized with compact cell structure," in *IEDM*. IEEE, 2016.
[8] M. Imani *et al.*, "Mpim: Multi-purpose in-memory processing using configurable resistive memory," in *ASP-DAC*. IEEE, 2017, pp. 757–763.
[9] A. Shafiee and other, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *43rd ISCA*, 2016.
[10] S. Angizi *et al.*, "Rimpa: A new reconfigurable dual-mode in-memory processing architecture with spin hall effect-driven domain wall motion device," in *ISVLSI*. IEEE, 2017.
[11] W. Kang *et al.*, "In-memory processing paradigm for bitwise logic operations in stt-mram," *IEEE TMAG*, 2017.
[12] C.-F. Pai *et al.*, "Spin transfer torque devices utilizing the giant spin hall effect of tungsten," *Applied Physics Letters*, 2012.
[13] X. Fong *et al.*, "Spin-transfer torque devices for logic and memory: Prospects and perspectives," *IEEE TCAD*, vol. 35, 2016.
[14] Z. He *et al.*, "A low power current-mode flash adc with spin hall effect based multi-threshold comparator," in *ISLPED*. ACM, 2016.
[15] H. Noguchi *et al.*, "Novel voltage controlled mram (vcm) with fast read/write circuits for ultra large last level cache," in *IEDM*, 2016.
[16] R. Zhang *et al.*, "A method of majority logic reduction for quantum cellular automata," *IEEE TNANO*, vol. 3, no. 4, pp. 443–450, 2004.
[17] (2011) Ncsu eda freepdk45. [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK45:Contents
[18] G. Panagopoulos *et al.*, "A framework for simulating hybrid mtj/cmos circuits: Atoms to system approach," in *DATE*, 2012.
[19] R. Zhang *et al.*, "Threshold network synthesis and optimization and its application to nanotechnologies," *IEEE TCAD*, vol. 24, 2005.
[20] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory," in *Emerging Memory Technologies*. Springer, 2014, pp. 15–50.
[21] K. Malbrain, "Byte-oriented-aes: a public domain byte-oriented implementation of aes in c," 2009.
[22] Y. Wang *et al.*, "Dw-aes: A domain-wall nanowire-based aes for high throughput and energy-efficient data encryption in non-volatile memory," *IEEE TIFS*, 2016.
[23] N. Binkert *et al.*, "The gem5 simulator," vol. 39, 2011.
[24] S. Li *et al.*, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*. ACM, 2009, pp. 469–480.
[25] A. Stillmaker *et al.*, "Toward more accurate scaling estimates of cmos circuits from 180 nm to 22 nm," *VLSI Computation Lab, ECE Department, University of California, Davis*, 2011.
[26] S. Mathew *et al.*, "340 mv–1.1 v, 289 gbps/w, 2090-gate nanoaes hardware accelerator with area-optimized encrypt/decrypt gf (2 4) 2 polynomials in 22 nm tri-gate cmos," *IEEE J. Solid-State Circuits*, vol. 50, no. 4, pp. 1048–1058, 2015.
[27] Z. Abid *et al.*, "Efficient cmol gate designs for cryptography applications," *IEEE transactions on nanotechnology*, vol. 8, pp. 315–321, 2009.